

NETCT Handling

Faster Net Connect Table searches

Thu, Jun 19, 2003

The Net Connect Table (NETCT) houses Acnet task names and UDP ports along with the message queue id needed for passing along communications to the next level. For example, when a datagram is received that is destined for the Acnet UDP port, the NETCT is searched to obtain the message queue id for passing a reference to the datagram to the Acnet task. When the Acnet task sees a datagram reference for a request message, it searches NETCT for the destination task name so it can pass it to the handler for the RETDAT protocol. This note explores how to speed up such NETCT table searches for PowerPC systems, since the table is currently in (slow-to-access) nonvolatile memory, being a component of the TRING table.

Consider maintaining a fast memory copy of the table. The contents of the table contain entries that include fields for the task name (or UDP port#), a message queue id, and optionally, a process id and event mask. One more word contains a counter that indicates activity or message reception for that entry. (This is used by code in LOOPAAUX.) Except for the counter, the fields in the entry are relatively static. Certain page applications might install a new entry that will be removed upon exit, but most applications do not affect the contents of this table. A local application might install a new entry, but local applications are typically installed at boot time, not usually changing during system operation.

For the PowerPC, searching a fast memory copy is almost free, as there is a maximum of 23 entries to search. By "almost free," we mean that the search time would pale compared to a single access to nonvolatile memory. In contrast, nothing would be gained in an IRM, as its access to nonvolatile memory access is no slower than its access to volatile memory.

(Another approach would be to use the name table to replace the search time by a table lookup involving a hash of the task name. This is not as good for the PowerPC, because the name table support includes checking for a match against the nonvolatile table at least once, assuming no collisions.)

The main code that searches the NETCT may be NCFIND. If we make that routine smarter, so it can check for the fast copy, it may take care of things for most needs.

But there is a simplifying fact. The NETCT does not need to be nonvolatile at all! It is created from scratch at system reset time. Many references to NETCT find it via NCTRPTR, a constant (0x2E) that is the offset into the TRING table where the 2-byte offset to the NETCT is found.

Suppose there is a low memory variable that is the address of the fast memory copy of NETCT, or NULL if none exists. Then all the code that refers to NETRPTR can be modified to check for that variable being non-NULL first. This will avoid even the nonvolatile reference to that 2-byte offset from the start of TRING. When there is a fast copy of NETCT, there does not also need to be a slow copy. It would be useful for diagnostics, however, for the fast copy to be located in a fixed area of memory.

Refs to NCTRPTR:

```
IPNodeN.a (lookup entry via portId)
NetIntIP.a (establishes 2-byte offset)
NetLayer.a (NetCnct, NetDcnt, NetQueue, NetCheck, NCTName)
UDPLayer.a (UDPCnct, VPortId, UDPCheck)
```

Refs to NETCT:

- IPNodeN.a
- NetIntIP.a
- SNAP.a
- UDPLayer.a

Refs to NCFIND:

- ANet.a
- ArcInt.a
- InsChain.a
- NetLayer.a
- SNAP.a
- UDPLayer.a

There are quite a few references in all. The new code should examine a new low memory variable that is set to the address of the NETCT table. This table would always exist at a fixed location. All users that access the contents of this table should be modified to check for the address of the table in the new low memory address. For an application that is separately compiled, it might check for the low memory, deriving the current logic if it is NETCT. This would allow such an application to be downloaded to an old system. Any new system will have a correct address of the fast NETCT; any old system will have NETCT in that low memory pointer variable. The new system code will not look for a NETCT in the TRING table, since it would always exist in fast memory.

The number of accesses to slow memory in the present scheme during a call to NCFIND is $(2 + 2 * N)$, where N is the number of entries in NETCT scanned before a match is found. Each access is about 1 μ s, so if N = 4 for RETDAT, it takes 10 μ s per call. But such calls only occur when a request is received. (For replies, a search is not needed, since the task id, which is the index into the NETCT table, is included in the Acnet header.) In the Linac PowerPC nodes, however, there is a constant search for SRMD in connection with 15 Hz SRM-based data collection, and SRMD is found after 6 entries are searched. This happens for every SRM at 15 Hz. A few nodes have as many as 6 SRMs, so this can add up to $6 * 14 = 84 \mu$ s per 15 Hz cycle. For each such occurrence, another pair of accesses is needed to increment the counter, bringing us up to 96 μ s. This is still not a very large part of 66 ms. Maybe the time wasted in such searches is not worth fixing.